Manual for the TI-95 Utilities Cartridge 18 October 2024

Robert AH Prins - robert.ah.prins@gmail.com

(C) Copyright Robert AH Prins, 1988-2024

Introduction

The TI-95 Utilities Cartridge is a piece of software that was created over a considerable period of time. I was originally intended as a collection of separate short utility routines to enhance the functionality of the TI-95, but in the course of time, the concept of a fully integrated set of routines became much more attractive.

As for the routines, they range from such trivialities as a true -1/0/+1 sign function to a horribly complex (to program, at least) back-up routine, but in my personal opinion they add significantly to the built-in functionality of the TI-95. In many ways, the Utilities Cartridge can be compared with the Master Library that used to come with the TI-59.

Because the Utilities Cartridge is a very complex piece of programming, there may still be bugs in it, so if you find any anomalies, please let me know so that I can resolve them.

And finally a very important warning: The TI-95 Utilities Cartridge main menu sequence contains six menus, but the last of these six, containing the <DOW>, <EXE>, <REX> and <ROM> functions will only show up when the TI-95 is in SYSTEM MODE. This is done on purpose, as the functions on this menu can potentially do a lot of damage when used by the average user. For that reason I suggest that you stay away from this menu until the moment that you really need it.

Running programs in the Utilities Cartridge

To execute a program in the Utilities Cartridge, press:

RUN then <UTI>.
At the SELECT FILE: prompt, choose <RUN>.

The TI-95 Menu Display will first show <OFF> <LRN> <BKU> and at the far right -->. Continue to press --> to scroll through the functions/programs in the module. Note that items in the Menu Display will be contained in < > other than --> in this manual.
The sets of menus displayed as --> is pressed are:

```
<OFF> <LRN> <BKU>       -->
<PGM> <REG> <FIL>       -->
<CAL> <DST> <SFF> <UCD> -->
<ENT> <MOD> <POL> <SGN> -->
<CHR> <MRX> <XP>        -->
<DOW> <EXE> <REX> <ROM> -->
<OFF> <LRN> <BKU>       -->             (Back to the first menu)
```

For example, to run the Speedy Factor Finder program to determine if 5001 is PRIME, from turn on, you would press:

```
RUN <UTI>
<RUN>
-->
-->
<SFF>  Display shows TI-95 SFF
5001
<1ST>
```

And the display would show f = 3.

Indicating the first prime factor of 5001 is 3. It is not prime!

The <OFF> function (SBA 50E)

Whereas most HP calculators save anything and everything when they are turned off and on again, the TI-95 unfortunately "forgets" quite a lot, such as

- the value in the display
- the value in the T-register
- the contents of the Alpha register
- the pending operations in progress

To avoid losing all of the above, I have written the <OFF> function. It can be used from the keyboard, but it is also programmable.

When used from the keyboard, it will save the entire machine state, which means that you can start a calculation, press <OFF> and ON again, followed by "=", to see the result, as if the TI-95 was never switched off.

However, <OFF> is much more useful when used in a program, as it not only saves all of the above, but also the state of the "run"-flags, which means that a program will continue to run after the TI-95 is switched back on.

The following two examples show the possibilities of using <OFF> from the keyboard and from within a running program.

Using <OFF> from the keyboard

Press 1 + 2 y^x 1 INV LN x~t pi FIX 7 ALPHA A B C D E ALPHA

and follow it by pressing OFF, the normal OFF and then ON, only to find out that you might just as well have pressed OFF without pressing the above keys. By now you may also start wondering why TI tells you that the TI-95 has Constant Memory...

Now repeat it, but instead of pressing OFF, now press RUN <UTI> <RUN> <OFF> and follow that with ON. If you have done it correctly, you will notice that the <OFF> <LRN> etc. menu is still in the display:

- pressing CE will re-display 3.1415927
- pressing OLD will re-display ABCDE
- pressing x~t will re-display 2.7182818
- pressing 3 = will display 9.0000000, the result of 1 + 2 y^x 3

in short, that is everything you entered is still present!

Using <OFF> in a program

Enter the following short program

0000 1 + SBA 226 0005 1 + SBA 226 0010 1 + SBA 226 0015 1 + SBA 226 0020 1 + SBA 226 0025 SBA 358 0028 GTO 0000

and start it by pressing RUN <PGM>. The result will be rather obvious, the TI-95 quickly counts to 5 and subsequently it turns itself off. Turning it back on will show the usual "TI-95 PROCALC".

Now change SBA 358 into SBA 50E. This time the TI-95 will continue its counting as soon as it is turned back on, demonstrating that <OFF> even preserves the state of the run-flags!

Note: To stop the TI-95 from turning itself off, press BREAK or HALT while turning it on!

The <LRN> function (SBA 510)

The menu displayed by pressing LEARN seems perfectly suitable, and in fact it is. Yet, once the user enters LEARN mode, he or she will at some moment discover that three functions - DFA, SHW and the alphabetic backarrow character - are not directly accessible and two others - GTO and SBR - do not accept the hexadecimal digits required to access application ROM cartridges.

To put this problem to an end, use can be made of the <LRN> option provided with the Utilities Cartridge. If <LRN> is pressed, a new menu is displayed, showing the familiar <1st>, <PC>, <END> and <ESC> above the F-keys.

Although this menu and its associated functions may seem rather familiar, they aren't and to see why you should press any of the options. Doing so will not only put the TI-95 into LEARN mode, but it will display yet another menu, with the F1...F5 keys labeled with <GTO>, <SBR>, <DFA>, <SHW> and -->. Pressing F5 will furthermore bring up another menu, this time with < < >, <CP>> and another --> above F5 to take us back to the first menu.

What can be done with all of the above? I hope the following examples will make it clear.

The <GTO> function (SBA 326)

One of the minor inconveniences of the normal (= INV GTL) GTO is the fact that it doesn't allow the hexadecimal digits 'A' through 'F' to be entered as part of the address, something that would be useful as it would allow you to jump directly in to an application ROM cartridge, instead of forcing you to use the rather awkward method of having to store the address in a register and following it by a GTO IND.

To avoid this problem, I decided to make the built-in GTO function available as a keyboard function, because it accepts hexadecimal addresses without any modification. In fact, this function and some of its neighbours, <SBR>, <SHW> and < < >, are the shortest routines in the entire Utilities Cartridge, taking up only 7 bytes each...

Using the <GTO> function is easy, just press <GTO> and follow it by up to four digits, either decimal or hexadecimal.

The <SBR> function (SBA 328)

Everything mentioned above about the <GTO> function can be repeated all over again for this function, so I suggest you reread the above, substituting <SBR> for <GTO>...

The <DFA> function (SBA 514)

Whereas the <GTO>, <SBR>, <SHW> and < < > functions simply use the built-in routines, the <DFA> function doesn't, because there is no built-in DFA function, at least not in such a way that it can be used from the keyboard. So, to create this function, yours truly had to come up with a little bit (78 bytes) of homebrewn code.

The <DFA> can be used to create stand-alone DFA Fx:aaa@nnnn instructions, but I mainly wrote it to allow programs to be delabelized, that is, I use it to change the address fields of existing DFA instructions after assembling and removing all labels from some programs (Assembling only speeds up a program, removing all labels can save considerable amounts of memory...)

- To use the <DFA> function to create DFA Fx:aaa@nnnn instructions, you should perform the following steps:
    - Find the DFN or DFA instruction that needs to be modified
    - use the arrow key to step past it

- get into INSERT mode
- enter a GTO instruction with the required address directly after the DFN/DFA
- use the backarrow key to step back to the DFN or DFA instruction
- press <DFA> and notice that
  - if there was a DFN instruction, it changes into DFA instruction
  - the label of the DFN instruction changes into the address previously following the GTO instruction OR
  - the old address of the DFA instruction changes into the address previously following the GTO instruction

Note: You can generate DFA instructions with hexadecimal addresses, but due to additional requirements, these CANNOT be used to directly access ROM cartridge-based routines.

The <SHW> function (SBA 3A6)

The <SHW> function is once again a simple one, at least from a programmer's point of view. It uses the built-in SHW routine, but by making this routine available on the keyboard, it adds some functionality, as <SHW> not only accepts a numeric field (0..8), but also IND, which would allow you to retrieve the statistics registers in a more versatile way, for example in a loop.

The < < > function (SBA 20E)

The team responsible for the design of the TI-95 made several mistakes. One of them was the fact that they "wasted" 96 keycodes to implement the alphabetic capability, the HP-41C uses a special code to indicate that the following codes should be interpreted as text, and the TI-95 design team should have used a similar approach. What makes this waste even more dramatic is the fact that one of these 96 alphabetic characters, the backarrow, cannot be created without resorting to unusual means.

In order to resolve this problem, you can use the < < > function. It enters the backarrow character in memory, and as an added advantage, it does so without the user having to enter ALPHA mode.

The <CP>> function (SBA 518)

The <CP>> function was suggested by Scott Garver. It is similar to the built-in CP function, but it has two advantages over the latter

- it asks for confirmation
- it can CP from the current step, not touching the steps before it

Using it is simple, step to the last step that you want to keep and press <CP>>. After having pressed <CP>>, the TI-95 will ask you to confirm your decision, and after pressing <YES>, it will perform the required partial CP. (Should you press <NO>, obviously nothing will be done).

The <BKU> function (SBA 51C)

The <BKU> function is the unsuccessful result of trying to convert the entire PCL program that comes with the PC-Interface into assembly language. Of course the function itself works and in some ways it's a nice addition to PCL, because PCL lacks an option to save the entire 7200-byte user memory on a PC.

After having pressed <BKU>, a new menu is displayed, offering three options, <GET>, <PUT> and <VFY>, used to read, save and verify a saved memory image and one <ESC> to leave the <BKU> function and to return to the main menu sequence.

As all three are very similar, I will only describe the first of them, <GET>.

The <GET> function (SBA 51E)

This function is used to save the entire 7200-byte user memory PLUS the 8-byte information in SR 2038 about the partitioning into a 7208-byte file on the PC.

After having pressed <GET>, the TI-95 asks for confirmation, and after confirming, by pressing <YES>, that this is the required function, it will enter data-entry mode, allowing you to enter a filename that satisfies the usual PC/MS-DOS requirements. At this stage it is still possible to abort the operation by pressing F5 <ESC>, and in order to make corrections, CLEAR can be used to remove the entered filename. However, upon pressing F4 <ENT>, the TI-95 will perform the requested operation, and when it finishes, it will give an appropriate message.

To summarize the flow,

1. Select the required function, in this case <GET>
2. Confirm that it is the required function by pressing F1, <YES> ¸ pressing F2, <NO> will return you to the main BKU menu
3. Enter a filename, and press F4, <ENT>
    - pressing F4, <ENT> without having entered a filename will generate a "No filename" error with errorcode 31(!) and return to the main BKU menu
    - pressing F5, <ESC> will abort the function and return to the main BKU menu
4. The operation will be performed, and after it finishes, control will return to the main BKU menu with a useful message. If an error occurs, an appropriate error message will be displayed, and although the error will most likely be an I/O error, the errorcode will once again be 31.

The <PUT> function (SBA 520)

The <PUT> function functions exactly like the <GET> function, but it obviously writes the memory out to a PC file.

The <VFY> function (SBA 522)

The <VFY> function functions exactly like the <GET> and <PUT> functions, but it obviously verifies the memory against a PC file.

The <PGM> function (SBA 526)

The <PGM> function isn't really a function, but rather it gives access to a second level menu, containing three functions that operate on programs,

- <CRC>, to calculate checksums
- <IAS>, a bug free version of INV ASM
- <LST>, an extension to the LL-function

The <CRC> function (SBA 528)

The <CRC> function was created to simplify checking the correct key-in of programs. I wrote it years ago, sent a copy to Palmer for inclusion in the Notes, but for some reason he never published it. The routine simply calculates a 2-byte checksum for the program in program memory, and if this checksum is equal to the one supplied by the author, you can be pretty sure that the program has been entered without mistakes.

The <IAS> function (SBA 52A)

The <IAS> function is an almost straight forward copy of the built-in INV ASM function, with one exception, it doesn't contain the bug that causes the label in some GTL, SBR or DFN instructions to be mutilated.

The <LST> function (SBA 52C)

The <LST> function once gives access to a deeper level menu, with three functions, <GTL>, <SBL> and <DFN>, and a more or less traditional <ESC> to return to the higher-level menu.

As all three functions are very similar, the even have the same SBA number, SBA 52E, I will only discuss the first of them, <GTL>.

The <GTL> function (SBA 52E)

The <GTL> function is very similar to the built-in LL-function. Whereas the built-in function produces a list of all labels in a program, the <GTL> function produces a list of all GTL instructions, or, when preceded by INV, a list of all GTO instructions.

Compared to the built-in LL-function, the <GTL> function has one little drawback, it always starts the listing at step 0000.

The <SBL> function (SBA 52E)

This function is equivalent to the <GTL> function, but it lists all SBL or SBR instructions.

The <DFN> function (SBA 52E)

This function is equivalent to the <GTL> and <SBL> function, but it lists all DFN or DFA instructions, including the DFN CLR and DFN Fx CLR ones.

The <REG> function (SBA 530)

The <REG> function isn't really a function, but rather it gives access to a second level menu, containing three functions that operate on registers:

- <CMS> to clear registers
- <LR> to list registers
- <SHL> to sort registers

All three of them use the general (= nnn.sss) register control word, to indicate the number of registers to be operated upon (= nnn) and the starting register (= sss).

The <CMS> function (SBA 532)

The <CMS> function is a versatile addition to the built-in CMS-function, as it allows a more selective clearing of registers, using a "nnn.sss" control value in the display to select the number (= nnn) of registers to clear and starting (= sss) register.

As the value of nnn.sss can be erroneous, <CMS> applies the following restrictions,

- if there are no registers, <CMS> will act as NOP
- if nnn.sss < 0 or nnn.sss > 1000, <CMS> will act as NOP
- if sss exceeds the number of the highest register, <CMS> will act as NOP
- if nnn + sss exceeds the number of the last register in the current partitioning, <CMS> will only clear the registers up to the last register in the current partitioning

The <LR> function (SBA 534)

The <LR> function is a versatile replacement for the built-in LR-function. It's advantages over LR are

- it can list a "nnn.sss" range of registers
- it can optionally skip registers containing zero
- it will add register numbers when in UNF mode

- it can produce 13d + exponent listings, but while doing so, it will assume that the requested registers contain valid decimal data...

To perform these tasks, it uses two flags, flag 16 and 17. The function of both of these flags is clarified in the table below.

Flag	Reset	Set
16	Skip if register = 0	Include all registers
17	Respect display mode	Print 13d + mantissa

The <SHL> function (SBA 534)

The <SHL> function is a sorter for registers containing numerical data. It uses the Shell sort algorithm, and it can sort up to 898 registers at blazing speed, sorting 750 random numbers takes a mere 21 seconds.

As was the case with <CMS> and <LR>, <SHL> expects a "nnn.sss" number in the display to tell it which registers to sort, and where to start sorting, with the same restrictions on "nnn.sss" as <CMS>, plus one extra, sss = 0 implies sss = 1, that is you cannot include register 0 in the sort.

As for errors, <SHL> will generate a "SHELLSORT ERROR", errorcode 31, in the following cases

- if there are no registers in the current partitioning
- if nnn.sss < 1
- if nnn is equal to the number of the last register in the current partitioning, that is the last register in the current partitioning cannot be included in the sort.

The <FIL> function (SBA 538)

The <FIL> function is, just as the <PGM> and <REG> functions merely an access function. Pressing it will put you on a menu with two useful functions,

- <MRF>, to merge a file from MEM file space to the program in PGM
- <REN>, to rename a file in MEM file space

The <MRF> function (SBA 53A)

A function sorely missing on the FILES menu is a function to merge files. The need for such a function may not be immediately clear, after all it is possible to use 'DIRFIL' RUN GTL/SBL xx etc. sequences to call external routines, but if "FIL" has a length of only 50 bytes and is called 6 times, it might be a lot more efficient to add "FIL" to the program...

Using <MRF> is simple, after pressing <MRF> the display will display the message "MERGE FILE " followed by a flashing cursor. You can now simply enter the name of the file to be merged.

If the merge is successful, a message "FILE xxx MERGED" will confirm it, otherwise you may see one of the following error messages:

- "INADEQUATE SPACE", in case there is not enough space in program memory to accommodate the file
- "INVALID DIR/FILE", in case the requested file doesn't exist

The <REN> function (SBA 53C)

Another function sorely missing on the FILES menu is a REName function, which makes renaming files very difficult - you need the STB function and a lot of calculations are involved to find the location of the filename to be changed. To simplify the process, I wrote <REN>, which allows you to rename a file in MEM file space, using a similar custom field-type instruction as the one used in <MRF>.

So, to rename a file, you simply press <REN>, which will bring a "RENAME " prompt in the display, followed by the flashing cursor. You can now enter the old filename. After entering a valid old filename, the display will change into "RENAME xxx - " with a flashing cursor after the "-", waiting for you to enter a new filename. After accepting it, a message, "FILE xxx RENAMED" will confirm the success of the operation.

As for possible errors, there is only one error, "INVALID DIR/FILE" which can appear

- if the old file doesn't exist or
- there is already a file with the same name as the new file

The <CAL> function (SBA 53E)

The <CAL> function is yet another function that is only used to access a next menu, this time to access two routines,

- <CJ> to convert a date to its JDN
- <JC> to convert a JDN to a date

Both of these routines are copies of the Roger Hill's equivalent routines in the HP PPC ROM, but during the conversion from HP-41ese to TMS 7000 assembler, I have added some useful extras to them.

The routines can be used to calculate the number of days between dates, a date in the future or past, given the current date and a number of days and the day of the week for a given date. They accept or produce dates using either the Julian or Gregorian calendar, and even the format, the European DDMM.YYYY or American MMDD.YYYY can be selected.

The relevant flag settings for these options are in the table below.

Flag	Reset	Set
16	MMDD.YYYY	DDMM.YYYY
17	Gregorian	Julian
18	Ext alpha	Std alpha (Only in conjunction with JC)

The formulae used in <CJ> and <JC> are:

<CJ>:

$$JDN = INT(INT(INT(367 * y') - INT(y') - .75 * INT(y') + DD) - .75 * \boxed{INT(y' / 100)}) + 1{,}721{,}115$$

where $y = YYYY + (MM - 2.85) / 12$

<JC>:

$$N = JDN - 1{,}721{,}119$$
$$C = INT((N - .2) / 36524.25)$$
$$N' = N + \boxed{C - INT(C / 4)}$$
$$Y' = INT((N' - .2) / 365.25)$$
$$N'' = N' - INT(365.25 * Y')$$
$$M' = INT((N'' - .5) / 30.6)$$
$$DD = INT(N'' - 30.6 * M' + .5)$$

If $M' < 9$: $YYYY = Y'$, $MM = M' + 3$

If $M' > 9$: $YYYY = Y' + 1$, $MM = M' - 9$

In both cases, the boxed expression must be replaced by 2 for the Julian calendar.

The <CJ> function (SBA 540)

The <CJ> function takes a date, after March 1, 0000, and converts it to its corresponding Julian Day Number, that is the number of days elapsed since January 1, 4713 BC (and don't ask me why astronomers selected that date!).

The result is returned in the display and the alpha register, with a "JDN=" descriptor.

A few examples

Date	16	17	Result	Comments
320.1960	-	-	JDN= 2437014.	Mar 20, 1960, US format, Gregorian
1004.1582	-	+	JDN= 2299160.	Oct 4, 1582, US format, Julian
1510.1582	+	-	JDN= 2299161.	Oct 15, 1582, EU format, Gregorian

The last two examples are the last date in the Julian calendar, and the first date in the Gregorian one, that is Pope Gregory XIII threw out October 5 through 14 to resynchronize the calendar with the earth’s revolution around the sun.

The <JC> function (SBA 540)

The <JC> function is the exact opposite of the <CJ> function, it takes a JDN and converts it back to a date. In addition to <JC>, it also uses flag 18 to optionally suppress the extended alpha mode.

A few examples

JDN	16	17	18	Result	Comments
2437014	-	-	-	Sun Mar 20, 1960	Mar 20, 1960, ext alpha (US = EU)
2437014	-	-	+	MDY= 3-20-1960	Mar 20, 1960, US format, std alpha
2437014	+	-	+	DMY= 20-03-1960	Mar 20, 1960, EU format, std alpha

The <DST> function (SBA 544)

The <DST> function is yet another function that is only used to access a next menu, and once again to access two routines,

- <NOR>, to calculate the normal distribution
- <INO>, to calculate the inverse normal distribution

Both routines use the standard Abromowitz and Stegun approximations.

The <NOR> function (SBA 546)

This function can be used to calculate Z(x) and Q(x) for the normal distribution given x.

An example

Suppose the lifetime of a lamp is normally distributed with a mean of 5000 hours and a standard deviation of 1000 hours. What percentage of the lamps will last 6500 hours?

Calculate x = (6500 - 5000) / 1000
Press <NOR>. The display will show "Q(x)=.0668072287", which means that about 6.7% of the lamps will last 6500 hours.

Note: The calculator will also calculate Z(x), which can be found in the T-register. Press x~t to view it.

The <INO> function (SBA 548)

This function performs the inverse of <NOR>, that is given Q(x), it calculates x.

An example

Consider the same lamp as in the example given with <NOR>. At some time during the testing only 1% is still surviving. Give an approximation of the duration of the test.

Enter Q(x), 0.01 and press <INO>. The display will show "x= 2.326785333". ¸ Multiply this number by the standard deviation (1500) and add the mean (1000) The result, 7326.78... gives the approximate duration of the test.

The <SFF> function (SBA 54A)

The problem of factoring numbers has always been a source of inspiration for programmers, nowadays maybe even more with the arrival of public key crypto- systems, which rely on the fact that it is (should be!) impossible to find the two 100-digit numbers that are used to generate the keys.

Obviously such a task cannot be handled by the TI-95, and certainly not by TI's absolutely disastrous <PF> function, which cannot even handle 13-digit numbers.

To overcome this limitation and the total utter lack of speed of the <PF> function, <SFF> was written. It is based on a modulo-30 sieve of Eratosthenes (<PF> uses a modulo-6 sieve). As might be expected, <SFF> can handle 13-digit numbers, and a second advantage is the fact that it contains a monitor-function, by pressing the [=]-key anytime during the factoring, it shows the current factor to be tested.

Also note that <SFF> is once again merely the front-end for the actual factoring routines, which are <1st> and <NXT>.

As for the speed, <SFF> tests about 54.5 factors per second...

The <1ST> function (SBA 54C)

The <1ST> function is used to find, as is obvious, the first factor of the number entered by the user.

The <NXT> function (SBA 54E)

The <NXT> function is used to find, just as obvious, all subsequent factors of the number entered by the user.

Some examples

1. Factor 987654321, one of the TI-59 benchmark numbers.
   - Enter 987654321 and press <1ST>. Almost immediately the display will show "f= 3." Press <NXT> 3 more times, to get the next three factors, 3, 17 and 17.
   - Press <NXT> a fourth time and immediately press the [=] key. You will see the factors being tested flash by in the windows above F3...F5. About three seconds later the display will show "f= 379721."
   - Press <NXT> again. The 1 that shows up indicates that all factors have been found.

2. Enter the following program:

   0000 SBA 54C PAU 1 IF= 2079 HLT SBA 54E GTO 0003

   and enter 103569859 (the second historical TI-59 test value) followed by pressing RUN <PGM>. A mere 5 seconds later a 1 in the display will indicate that the number

has been factored, but don't forget that 3 of those 5 seconds were used in the three PAU instructions.

The <UCD> function (SBA 550)

Computers don't make mistakes, but their human operators do, certainly when there are only a few millimeters between the key used to CD MEM file space and the key needed to CD cartridge file space.

To help those poor souls, and the ones who create assembly language routines and afterwards execute them as if they were normal programs, without realizing that hexadecimal 1F also means CD, <UCD> was developed.

The <UCD> function gives access to an UnCD menu, but unlike many of the other functions that give access to other menus, it also performs some preliminary work to set up the structures that are needed to restore the directory of a CDed MEM file space. (Note that <UCD> can only UnCD MEM file space!)

Once you have pressed <UCD>, the TI-95 will perform these setups, momentarily displaying the message "TI-95 UnCD", before halting with the question "Add file xxx?", where "xxx" is the name of the first file in the CDed file space. At this stage, the sub-menu will also be above the F-keys. It contains two options,

- <YES>, to indicate that the current file should be added
- <END>, to indicate that the UnCD process is complete and to transfer the temporary directory to the real directory

Of course <ESC> is also present, to abort the entire UnCD process.

The <YES> function (SBA 552)

The <YES> function is used to update the temporary directory with the data obtained from the to-be-restored file.

The <END> function (SBA 554)

The <END> function is used to transfer the temporary directory to the actual directory at the top of MEM file space.

Note: If the TI-95 encounters erroneous filenames, that is filenames with non- ASCII characters, it will automatically perform the <END> function. If this is not desired, the non-ASCII test can be bypassed by putting the calculator in SYS mode.

The <ENT> function (SBA 556)

The <ENT> function provides a mathematical INT function, that is it reduces the number X it operates upon to the smallest integer less than or equal to X.

Examples
123.456 <ENT> will result in 123
123 <ENT> will result in 123
-123 <ENT> will result in -123
-123.456 <ENT> will result in -124

The <MOD> function (SBA 558)

The <MOD> function operates on two numbers, X and Y. It returns two other numbers, Q and R. The relation between X, Y, Q and R is defined in the following two equations:

1. X = Y * Q + R, where Q is any integer
2. 0 <= ABS(R) < ABS(Y)

Examples

The following table lists the results, Q, the t-register and R, the display, of applying the <MOD> function on X, the display and Y, the t-register.

X	Y	Q	R
12.34	56.78	0.	12.34
-12.34	56.78	-1.	44.44
12.34	-56.78	0.	12.44
-12.34	-56.78	1.	44.44
56.78	12.34	4.	7.42
-56.78	12.34	-5.	4.92
56.78	-12.34	-4.	7.42
-56.78	-12.34	5.	4.92
0.00	12.34	0.	0.00
0.00	-12.34	0.	0.00
12.34	0.00	+++	12.34
-12.34	0.00	---	-12.34

Note:

--- = -9.999999999999 99
+++ = 9.999999999999 99

The equation X = Y * Q + R applies to all but the last two rows in the table.

The <POL> function (SBA 55A)

Although <POL> appears on the main menu sequence, it should really only be used through its keystroke coded counterpart. It evaluates a polynomial of the format f(x) = A(n)x^n + A(n-1)x^(n-1) + ... A(1)x + A(0), but as it uses one of the two internal polynomial evaluation routines, it requires the An to be stored in internal floating point format and in specific system registers. To put them into those registers, use should be made of the keystroke <POL> program. The instructions for this program are:

1. Press RUN <UTI> --> <POL>. The display will briefly show the message "POLYNOMIAL EVAL", to halt with "ENTER DEGREE", so
2. Enter the degree and press < N >. The display will show A(0), so
3. Enter A(0) and press <ENT>. Repeat this step for A[1] through A[n]. After entry of A(n), F1 will change into <CAL>, while <ESC> will appear above F5.
4. Enter x, and press <CAL>. On the display Fx will show.

Example

Calculate Fx for the polynomial 4 * X ** 3 + 3 * X ** 2 + 2 * X + 1.

1. Press RUN <UTI> --> <POL>
2. Enter 3 and press < N >
3. Enter the four A(i), starting with A(0)
4. Enter 10 and press <CAL>. The resulting display will be "Fx= 4321."
5. Enter 3.1415 (PI) and press <CAL>.
   This time the display should show "Fx= 160.9171052".

The <SGN> function (SBA 55C)

The <SGN> function returns the mathematical sign of the number in the display.

Examples

The following list gives the returned values:

- Number in the display < 0: <SGN> will return -1
- Number in the display = 0: <SGN> will return 0
- Number in the display > 0: <SGN> will return 1

The <CHR> function (SBA 55E)

The <CHR> function can be used to change the shape of CHR 000 through CHR 004, using the fact that the TI-95 uses a programmable Hitachi LCD controller chip. Input to the <CHR> function is a 16-digit unformatted number that describes which character should be re-shaped and its new shape.

The 16-digit unformatted number represents 8 bytes, and each of them can have a value in the range 00 to FF, but as the rows in the display are made up of only 5 pixels, the three most significant bits of the bytes are not used, effectively reducing the range from 00 to 1F, or in binary, from 00000 to 11111.

By now it may be obvious (or not?) that the 5 useable bits in the 8 bytes in the unformatted number determine the status of the 5 pixels in the 8 rows of each character in the display! The first version of <CHR> used this similarity, using the T-register to store the desired pattern and the display to select the character to modify. However, quite soon it became clear that the three bits remaining in the first byte could be used to store the character to be modified, the method still used in the currently implemented version of <CHR>.

During the development it became also clear that, although the Hitachi chip used to control the LCD supports 8 user-defined characters:

- CHR 005...007 cannot be used, because CHR 005 is used internally to replace the standard "÷" character by a "\"
- CHR 006 is used internally to replace the standard arrow character by a "~"
- CHR 007 is used internally to map the status of the various indicators

This last feature can be verified by executing the following steps:

1. Enter alpha mode
2. Press --> followed by CHR 007
3. Press <LC>, 2nd or --> <INS> and notice the changes, leave alpha mode and press INV or select another angular mode or base.

Examples

In the following example, CHR 000...004 are re-defined as
CHR 000: a forward arrow
CHR 001: garbage
CHR 002: a car on the street
CHR 003: a power of two character
CHR 004: a box

To do so, these characters are first drawn in a 5 x 8 box, representing ON pixels by a * and OFF pixels by a " ", because that happens to produce be more readable results than a use of a 0 and 1.

```
CHR 000     CHR 001     CHR 002     CHR 003     CHR 004

      00    *   *  11    *****  1F     **   06   *****  1F
  *   04        *  01           00    *  *  09   *   *  11
   *  02    **  *  19    *  *   12      *   02   * * *  15
***** 1F      **   06    *****  1F     *    04   * * *  15
   *  02    *  **  13    *  *   12    ****  0F   * * *  15
  *   04    **  *  19           00          00   *   *  11
      00           00    *****  1F          00   *****  1F
      00    * **   16           00          00          00
```

Next the information to determine the character to be modified is added to the first three bits of each character, which results in the following:

CHR 000: 00
CHR 001: ..1 1...1 31
CHR 002: .1. 11111 5F
CHR 003: .11 ..11. 66
CHR 004: 1.. 11111 9F

so to create all of the above, the following steps should be executed:

1. Store Pi * 1 EE 65 in register A.
2. Select UNFormatted mode
3. Enter 0 0 0 4 0 2 1 Fh 0 2 0 4
4. Press <CHR>
5. Press RCL A
6. Press <CHR>
7. Enter 5 Fh 0 0 1 2 1 Fh 1 2 0 0 1 Fh
8. Press <CHR>
9. Enter 6 6 0 9 0 2 0 4 0 Fh
10. Press <CHR>
11. Press <CHR>
12. Enter 6 6 0 9 0 2 0 4 0 Fh
13. Press <CHR>
14. Enter 9 Fh 1 1 1 5 1 5 1 5 1 1 1 Fh
15. Press <CHR>

Note 1: Step 6 above functions, because 3101190613190016 and Pi * 1e65 have the same binary representation when the three MS bits of each byte are discarded!

Note 2: To show the current definition of all user defined characters without modifying them, an unformatted number with a first digit of Ah..Fh can be used.

The <MRX> function (SBA 560)

The TI-95 can display one and the same number in many formats. In many ways this is a very desirable characteristic, but sometimes it can be very annoying to constantly change modes to print data. It would be very nice if an instruction existed that would combine MRG and FIX without actually altering the current display mode.

Once again the Utilities Cartridge provides such a function, and it is so short, only 43(!) bytes in assembly language, that one really wonders why TI Inc needs no less than 48 bytes of slow keystroke code to accomplish something similar...

Examples

Example 1

Although keyboard use of the <MRX> function is perfectly well possible, it will mostly be used in programs. However, this first example may give an indication of its usefulness:

1. Put the TI-95 in unformatted mode.
2. Put "PI=" in the Alpha register.
3. Press COL 16 and leave Alpha mode.
4. Press <MRX> 5 =. The display will show "PI= 3.14159", but when you press CE, you will notice that the TI-95 is still in unformatted mode!

Example 2

Suppose someone in computer science always uses his or her TI-95 in HEX mode, but in a few programs some data needs to be printed in FIX n DEC mode. Without using the Utilities Cartridge, he or she would have to resort to the program in the left column. With the Utilities Cartridge in place it could be done with the program in the right column:

Program without Util Cart		Program with Util Cart
.		.
.		.
HEX	0	HEX
.		.
.		.
'Address 1=' COL 16	0	'Address 1=' COL 16
MRG A PRT	0	MRG A PRT
COL 21	0	COL 21
'Value 1=' COL 36	0	'Value 1=' COL 36
DEC FIX 2	0	SBA 53E
OLD	+1	FIX 2
MRG B PRT COL 41	0	MRG B PRT COL 41
HEX OLD	-2	
'Address 2=' COL 56	0	'Address 2=' COL 56
MRG C PRT	0	MRG C PRT
COL 61	0	COL 61
'Value 2=' COL 76	0	'Value 2=' COL 76
DEC FIX 4	0	SBA 53E
OLD	+1	FIX 4
MRG Q PRT	0	MRG Q PRT
HEX	-1	.
.		.
.		.

In this case the difference may not be very striking, the program on the right is just one byte shorter than the program on the left, but the advantage is that the user knows that the actual display mode does not change!

Note: When <MRX> is used in a program it must always be followed by the normal FIX and MRG instructions, although <MRX> only uses the field-part of these instructions.

The <XP> function (SBA 562)

The <XP> function is yet another menu-access function. Pressing it will put you on the "EXT PRECISION" menu, where the following two functions can be found,

- <SQR>, to calculate roots up to 2550 digits
- <X!>, to calculate factorials up to 1000!

Obviously the use for these functions is very limited, I just included them because I happen to love extended precision programs... Besides that, it was an interesting challenge to translate to original TI-59 keystroke coded programs into TMS 7000 assembly language.

The <SQR> function (SBA 564)

The <SQR> function is an extended precision square root function, that is capable of calculating roots of numbers X, with 0 < X < 100.

<SQR> expects two numbers as input,

- the number of which the root has to be found. This number should be in the display.
- the number of 10-digit blocks. This number should be stored in register 0000.

However, it should be noted that <SQR> may change the number of 10-digit blocks to a mere 10, if it detects that it doesn't have enough registers to accommodate twice this number of registers, it detects a negative number of blocks

It should also be noted that <SQR> prints the 10-digit blocks as they are found, but that reprinting them is difficult, as twice the value of them is stored in registers 0001 through N, with N the number of requested blocks.

For example, to compute the square root of 2 to 50 places, press

5 STO 0000
2 <SQR>

The display will show the following values:

1414213562
3730950488
0168872420
9698078569
6718753769

The <X!> function (SBA 566)

The <X!> function is an extended precision X! function, that is capable of calculating all digits of up to 1000! It is basically just as useless as <SQR>.

It also has a serious shortcoming, because cannot repartition itself. When it runs out of space, it does the next best thing, it clears the registers it has so diligently filled with an intermediate result, and subsequently stops with an "INVALID REGISTER" error.

Note: To calculate 1000! you need 258 registers, and when the calculations have finished, after about 15'30", the first 8 digits of 1000! can be found in register 257...

For example, to compute all the digits of 69!, press 69 and <X!>. When the program finishes running (which is nearly instantly), the digits will be found in registers 10 and lower.

Press RCL 0010 to see 171122452.
Press RCL 0009 to see 4281413113.
Press RCL 0008 to see 7246833888.
Press RCL 0007 to see 1272839092.
Press RCL 0006 to see 2705448935.
Press RCL 0005 to see 2036939364.
Press RCL 0004 to see 8040923257.
Press RCL 0003 to see 2797541406.
Press RCL 0002 to see 4742400000.
Press RCL 0001 to see 0000000000.

The <DOW> function (SBA 568)

The <DOW> function allows the creation of copies of the <REN>, <MRF> and <ROM> routines in MEM file space and system registers 114...139. These routines can subsequently be used to act upon the cartridge containing the Utilities Cartridge or any other RAM cartridge, with one exception, the version of <ROM> created by <DOW> can also act on ROM cartridges.

Because the actual code that makes up these three routines resides in system-registers that are also quite heavily used by the Utilities Cartridge, I have made "+UC" a once-only program. If it is needed a second time, you will have to download it again!

Furthermore, I very strongly recommend that these routines are only used by following exactly these steps:

1. Press <DOW> Executing <DOW> will
   - delete the file "+UC", if it can find a file with this name in MEM file space.
   - check if the free space in MEM file space is at least 16 bytes. If the space is less than 16 bytes, it will generate an "INADEQUATE SPACE" error.
   - put the next menu above the F1...F5 keys.

2. Select the required routine (or <ESC>), and press the corresponding Fx key. Selecting any of the three routines will put the message "FILE +UC CREATED" in the display.

3. If "+UC" was downloaded to act upon another cartridge than the one containing the UC, switch of the TI-95 and insert the required other cartridge.

4. Put 'MEM+UC' in the alpha register.

5. Press FUNC <SYS> <YES> to put the TI-95 in SYSTEM MODE.

6. Optional, and only needed when you're using the downloaded version of <ROM>,
   - Select UNFORMATTED MODE
   - Put the required downloading information in the display (See the description of <ROM> later in this manual for the required format)

7. Press SBA 600 to perform the required operation.

As for the descriptions of the downloadable versions of <MRF> and <REN>, see the descriptions of <MRF> and <REN> on prior pages, but remember that the downloaded "+UC" callable versions only operate on file space in RAM cartridges, and not on MEM file space!

And concerning the downloadable version of <ROM>, it functions exactly the same as the original <ROM>, described later in this manual.

The <EXE> function (SBA 56C)

The <EXE> function can be used to execute SBA 6-- routines in MEM File Space, without having to put "MEMFIL" in the alpha register. An added bonus is the fact that it even allows calling "PGM" as if it were an assembly language routine, which makes it very easy to test SBA 6-- routines that are still under development, because it removes the necessity of constantly having to save them in file space.

Note: The function was mainly included for personal use of yours truly, and as such I only recommend that it is only used by very experienced TMS 7000 assembly language programmers.

The <REX> function (SBA 56E)

The <REX> function, in combination with the information that can be found in the files containing the disassembled TI-95 system keystroke coded functions, allows calling parts of these routines as subroutines in normal user programs.

An example

To display the error message "OVERFLOW", the following sequence can be used:

1. Press <REX>. The display will show a "ROM EXECUTE" prompt, followed by the cursor.
2. Press 0 7 Ah 1.

Note: When used in a program, SBA 56E should be followed by a GTO instruction. The <GTO> function on the LEARN menu can be used to create addresses with hexadecimal digits.

The <ROM> function (SBA 570)

The <ROM> function can be used to transfer any block of 4Kb of memory from the total address space of the processor into user memory. Its main use is to allow users to have a look at the keystroke programmed internal functions and ROM cartridges, but hackers may use it to have a look at the assembly language coded parts of the system ROMS.

The input for <ROM> is an unformatted number. The first digit should be 0, the second digit can be 0...3. This digit determines which page of the ROM should be accessed. The third through sixth digit constitute the address where the downloading (of 4K!) should start. Useful numbers are:

- 00C7A1: Start of keystroke coded functions on page 0
- 00D7A1: Continued keystroke code from page 0. Last valid steps are 2141 and 2142, containing the internal checksum for page 0.
- 01CC00: Start of keystroke coded functions on page 1.
- 01DC00: Continued keystroke code from page 1. Last valid steps are 1022 and 1023, containing the internal checksum for page 1.

Note: The addresses mentioned above, C7A1 and CC00 are real addresses, however, the ROM keystroke interpreter translates them into 07A1 and 2C00!

The Non-Menu functions in the TI-95 Utilities Cartridge

The functions described on the previous pages are all accessible through the main menu sequence. However, the Utilities Cartridge contains a number of functions that are so so "program-only" that it seemed rather wasteful to create two extra menus for them.

The <POP> function (SBA 572)

This is a programmer's only function. It removes pending returns from the subroutine stack, effectively changing a SBR into a GTO. Programmers might find use for it.

Example
```
0000 SBR 0003
0003 GTO 0000
```

The above program will stop very quickly with a "SBR STACK FULL" error, but by adding a SBA 572 between the SBR and GTO, it will run forever.

The <R/B> function (SBA 574)

The <R/B> function emulates the TI-59 RTN function, that is it acts as RTN when it is encountered in a subroutine and the subroutine return stack is not empty and it acts as BRK when there are no pending return addresses

Its main use is that it allows a subroutine also to be used as a restartable stand-alone routine.

Example: The following program can be used to calculate both factorials and e to the power of x, although this last function isn't exactly very accurate...

It is based on the fact that e-to-the-power-x equals 1 + x/1! + x/2! ...

```
0000 DFN F1:x! @01 DFN F2:EXP@02 RTN

0015 LBL 01
0018   STO A (
0021   RCL A * DSZ A GTO 0021 1 )
0031   SBA 574
0034   GTL 01

0037 LBL 02
0040   STO B
0042   10 STO C (
0048   RCL B y^x RCL C / RCL C SBL 01 + DSZ C GTO 0048 1 ) HLT
```

The <RPO> function (SBA 576)

The <RPO> function can be used to restore the contents of the pending operations stack previously saved by <STO>. It is especially useful when a BRK is used during pending operations.

Example

Enter the following program

```
0000 ( 1 + 2 * 3 y^x SBA 57A BRK SBA 576 4 ) HLT
```

then

1. Press RUN <PGM> CLEAR
2. Enter 9 y^x 8 * 7 + 6 =, to see 301327053 appear
3. Press <GO> and see 163, the result of (1 + 2 * 3 y^x 4)

The <SBR> function (SBA 578)

The <SBR> function is mainly intended as an extension to the error handler. It provides a neat print of all return addresses stored in the subroutine stack when a 'SBR STACK FULL' error occurs.

The format of the printout is:

```
   DIR FILE STEP      DIR : Directory to return to
1: xxx  xxx xxxx      FILE: File to return to, blank if DIR is 'PGM'
.                     STEP: Return address
.
.
8: xxx  xxx xxxx
```

If a printout is required when a program has been temporarily been stopped, it can only be obtained by pressing FUNC <SYS> <SBA> 578, in other words, DO NOT USE RUN <UTI> -->... <SBR>, because pressing RUN at any moment will reset the subroutine stack!

The <SPO> function (SBA 57A)

The <SPO> function can be used to save the contents of the pending operations stack, so that they can later be restored by <RPO>. It is especially useful when a BRK is used during pending operations.

Example

Enter the following program

```
0000 ( 1 + 2 * 3 y^x SBA 57A BRK SBA 576 4 ) HLT
```

then

1. Press RUN <PGM> CLEAR
2. Enter 9 y^x 8 * 7 + 6 =, to see 301327053 appear
3. Press <GO> and see 163, the result of (1 + 2 * 3 y^x 4)

The <STU> function (SBA 57C)

The <STU> function can be used to transfer the contents of the system alpha register to the user alpha register. It is especially useful if results of the built-in functions, or results generated by the Utilities Cartridge have to be embedded in user-defined messages.

When used in a program, it can optionally be followed by COL nn, to indicate the starting column in the user alpha register for the transfer.

Example

The following program uses the <STU> function to transfer the extended alpha result from the <JC> function to the user alpha register:

```
0000 'Enter date' BRK 0011 CE 'That is ' 0020 SBA 540 SBA 542 SBA 57C COL 09 HLT
```

Appendix A: Function versus SBA

Functions on the first menu of the TI-95 Utilities Cartridge

Function	SBA	Notes
OFF	50E	Programmable
LRN	510	Not programmable
LRN-1st	512	Not programmable
PC	512	Not programmable
END	512	Not programmable
ESC	50C	Only mentioned for completeness' sake
LRN-...-GTO	326	Programmable
LRN-...-SBR	328	Programmable
LRN-...-DFA	514	Not programmable, use restricted to Learn mode only
LRN-...-SHW	3A6	Programmable
LRN-...--->	516	Not programmable
LRN-...- <	20E	Programmable, should not be used!
LRN-...-CP>	518	Not programmable, use restricted to Learn mode only
LRN-...--->	51A	Not programmable
BKU	51C	Not programmable
BKU-GET	51E	Not programmable
GET-YES	51E	Not programmable
YES-ENT		Not implemented as SBA
ESC		Not implemented as SBA
GET- NO	51C	Not programmable
BKU-PUT	520	Not programmable
PUT-YES	520	Not programmable
YES-ENT		Not implemented as SBA
ESC		Not implemented as SBA
PUT- NO	51C	Not programmable
BKU-VFY	522	Not programmable
VFY-YES	522	Not programmable
YES-ENT		Not implemented as SBA
ESC		Not implemented as SBA
VFY- NO	51C	Not programmable
BKU-ESC	524	Not programmable
-->	50A	Programmable, but useless(?)

Functions on the second menu of the TI-95 Utilities Cartridge

Function	SBA	Notes
PGM	526	Programmable, if followed by HLT, it will display the "PGM UTILITIES" menu.
PGM-CRC	528	Programmable
PGM-IAS	52A	Programmable
PGM-LST	52C	Not programmable
LST-GTL	52E	Not programmable
LST-INV-GTL	52E	Not programmable
LST-SBL	52E	Not programmable
LST-INV-SBL	52E	Not programmable
LST-DFN	52E	Not programmable
LST-INV-DFN	52E	Not programmable
LST-ESC	526	Programmable, if followed by HLT, it will display the "PGM UTILITIES" menu.
REG	530	Programmable, if followed by HLT, it will display the "REG FUNCTIONS" menu.
REG-CMS	532	Programmable
REG-LR	534	Programmable
REG-SHL	536	Programmable
REG-ESC	50C	Only mentioned for completeness' sake
FIL	538	Programmable, if followed by HLT, it will display the "FILE OPERATIONS" menu.
FIL-MRF	53A	Programmable, should be followed by 3 alpha characters, containing the name of the file to be merged. This name MUST include trailing and leading blanks!
FIL-REN	53C	Programmable, should be followed by 2*3 alpha characters, containing the old and new filenames. The names MUST include trailing and leading blanks!
FIL-ESC	50C	Only mentioned for completeness' sake
-->	50A	Programmable, but useless(?)

Functions on the third menu of the TI-95 Utilities Cartridge

Function	SBA	Notes
CAL	53E	Programmable, if followed by HLT, it will display the "CALENDAR CALCS" menu.
CAL- CJ	540	Programmable
JC	542	Programmable
ESC	50C	Only mentioned for completeness' sake
DST	544	Programmable, if followed by HLT, it will display the "DISTRIBUTIONS" menu.
DST-NOR	546	Programmable
INO	548	Programmable
ESC	50C	Only mentioned for completeness' sake
SFF	54A	Programmable, if followed by HLT, it will display the "TI-95 SFF" menu.
SFF-1st	54C	Programmable
NXT	54E	Programmable
ESC	50C	Only mentioned for completeness' sake
UCD	550	Not programmable
UCD-YES	552	Not programmable
END	554	Not programmable
ESC	50C	Only mentioned for completeness' sake
-->	50A	Programmable, but useless(?)

Functions on the fourth menu of the TI-95 Utilities Cartridge

Function	SBA	Notes
ENT	556	Programmable
MOD	558	Programmable
POL	55A	Programmable, preferably only in conjunction with keystroke POL!
SGN	55C	Programmable
-->	50A	Programmable, but useless(?)

Functions on the fifth menu of the TI-95 Utilities Cartridge

Function	SBA	Notes
CHR	55E	Programmable
MRX	560	Programmable, should be followed by real FIX and MRG instructions.
XP	562	Programmable, if followed by HLT, it will display the "EXT PRECISION" menu.
XP -SQR	564	Programmable
X!	566	Programmable
ESC	50C	Only mentioned for completeness' sake
-->	50A	Programmable, but useless(?)

Functions on the sixth menu of the TI-95 Utilities Cartridge

Function	SBA	Notes
DOW	568	Not programmable, keyboard use only available in SYS mode
DOW-REN	56A	Downloaded <+UC> routine is programmable
MRF	56A	Downloaded <+UC> routine is programmable
ROM	56A	Downloaded <+UC> routine is programmable
ESC	50C	Only mentioned for completeness' sake
EXE	56C	Programmable, should be followed by 3 alpha characters, containing the name of the file to be executed. Keyboard use only available in SYS mode.
REX	56E	Programmable, should be followed by GTO instruction containing the address where to enter the ROM function. Keyboard use only available in SYS mode.
ROM	570	Not programmable, keyboard use only available in SYS mode. Requires input in UNFormatted mode.
-->	50A	Programmable, but useless(?)

Non-menu functions of the TI-95 Utilities Cartridge

Function	SBA	Notes
Start of UC	508	Programmable
POP	572	Programmable, not useable from the keyboard
R/B	574	Programmable, not useable from the keyboard
RPO	576	Programmable
SBR	578	Programmable, virtually unuseable from the keyboard, routine will automatically be called upon a 'SBR STACK FULL' error.
SPO	57A	Programmable
STU	57C	Programmable, virtually unuseable from the keyboard, can optionally be followed by COL nn to indicate the position where the transfer should begin.

Appendix B: Using the field functions in program

The Utilities Cartridge contains five functions that require one or more fields to function,

- <MRF>, merge file
- <REN>, rename a file
- <MRX>, merge in fix
- <EXE>, execute an SBA 6-- routine
- <REX>, access the keystroke coded internal ROM functions
- <STU>, transfer system- to user alpha register that may optionally be followed by a field.

Using these functions from the keyboard is simple, you just have to fill in the fields. However, using them in a program requires two different methods, because of the difficulties in entering the required fields.

The methods are:

- to use <MRF> in a program, use SBA 53A `FIL` (or `MEM+UC` SBA 600 `FIL` for the <DOW> created version) `FIL` must contain leading and trailing blanks, if it doesn't, you will end up with an "INVALID DIR/FILE" error.
- to use <REN> in a program, use SBA 53C `OLDNEW` (or `MEM+UC` SBA 600 `OLDNEW` for the <DOW> created version)

Both `OLD` and `NEW` must contain leading and trailing blanks. If you don't include leading or trailing blanks in `OLD` the result will most likely be an "INVALID DIR/FILE" error.

However, not including leading or trailing blanks in `NEW` can create 'protected' files, that is files that cannot be handled by the built-in file-handling functions.

Example

Enter this program,

```
0000 CE 'A TES!' HLT
```

and save it in MEM file space, giving it the name 'QXW'. Now CP program memory and enter this program,

```
0000 SBA 53C `QXWWX` RCL Z HLT
```

and RUN it. Quite surprisingly the program will stop with 'Z' in the alpha register, and if you run a FILES <CAT> <MEM>, you will discover that you now have a program with the name `WX‡`, which you can run from the keyboard, but which you cannot GET or DF anymore.

To "unprotect" it, use the following program,

```
0000 SBA 53C `WX` RCL Q `XW` HLT
```

- to use <MRX> in a program, use SBA 560 FIX [IND] a/nnn MRG [IND] a/nnn/=
- to use <EXE> in a program, use SBA 56C `FIL`, but you must make sure that 'FIL' is a valid SBA 600 (= TMS 7000 assembly language) routine.

  Warning:

  Using the <EXE> function to execute normal (= keystroke coded) routines in MEM file space will inevitably lead to partial or total loss of memory, including that of a RAM cartridge, so DON'T experiment!

- to use <REX> in a program, use SBA 56E GTO nnnn, where the nnnn should be a valid address in the ranges

  ```
  07A1 through 1FFB
  2C00 through 3FFB
  ```

  Warning:

  Using <REX> with addresses outside of these ranges may create serious problems, and may lead to partial or total loss of memory, including that of a RAM cartridge, so DON'T experiment!

- to use <STU> in a program, use SBA 57C and optionally follow it by COL nn. If COL nn is omitted, the transfer will take place to COL 01 of the user alpha register.